Impacto del patrón modelo vista controlador (MVC) en la seguridad, interoperabilidad y usabilidad de un sistema informático durante su ciclo de vida.

Influence of the model view controller (MVC) pattern on the security, interoperability and usability of a computer-system's life cycle.

Franklin Enríquez, Santiago Fierro, Brandon Flores , Daisy Imbaquingo, Jaime Michelena
Carrera de Software, Universidad Técnica del Norte, Imbabura-Ibarra, Ecuador
Corresponding author: deimbaquingo@utn.edu.ec

Vol. 02. Issue 01 (2023): July
DOI: 10.53591/easi.v2i1.2043
ISSN 29536634
Submitted: March 19.2023
Revised: June 02, 2023
Accepted: June 27, 2023

Engineering and Applied
Sciences in Industry
University of Guayaquil. Ecuador
Frequency/Year: 2
Web:
revistas.ug.edu.ec/index.php/easi
Email:
easi-publication.industrial@ug.edu.ec

How to cite this article:
Enríquez, F. et al. (2023). Impacto del patrón modelo vista controlador (MVC) en la seguridad, interoperabilidad y usabilidad de un sistema informático durante su ciclo de vida. *EASI: Engineering and Applied Sciences in Industry*, 2(1), 11-16. https://doi.org/10. 10.53591/easi.v2i1.2043

Resumen. Aplicar un patrón de diseño durante el desarrollo de software es una estrategia habitual entre los programadores. Dentro de la amplia clasificación de patrones de diseño creados hasta la fecha, se encuentran los patrones arquitectónicos, siendo uno de los más utilizados el patrón Modelo-Vista-Controlador, también conocido como MVC. La amplia adopción de este patrón ha hecho necesaria una evaluación de sus impactos en la seguridad, interoperabilidad y usabilidad a lo largo del ciclo de vida de un sistema informático. Mediante una revisión bibliográfica y una investigación en sitios web especializados en desarrollo de software, este estudio revela que el patrón MVC influye positivamente en estos aspectos. Además, se constató que la comunidad de desarrolladores experimentados considera indispensable y ampliamente popular el uso de este patrón de diseño arquitectónico.

Palabras claves: Arquitectura, Patrón, Diseño , Seguridad, Interoperabilidad, Usabilidad

Abstract. Applying a design pattern during software development is a common strategy among programmers. In the broad classification of design patterns created to date, there are architectural patterns, one of the most widely used being the Model-View-Controller pattern, also known as MVC. The wide adoption of this pattern has made it necessary to evaluate its impact on security, interoperability and usability throughout the life cycle of a computer system. Through a literature review and research on specialized software development websites, this study reveals that the MVC pattern has a positive influence on these aspects. Furthermore, it was found that the use of this architectural design pattern is considered indispensable and widely popular by the community of experienced developers.

Keywords: Architecture, Design, Pattern, Security, Interoperability, Usability.

1. INTRODUCCIÓN

Para el desarrollo de un sistema de software, los desarrolladores primero toman en cuenta las características esenciales y los beneficios que aportará éste durante su funcionamiento. El documento de especificación de requisitos de software ERS tiene como objetivo, documentar de manera clara y precisa el funcionamiento y las características esenciales que un sistema de software debe tener, para satisfacer las necesidades de los usuarios y desarrolladores. El diseño de software se lo define como, "el proceso que traduce los requisitos en un diseño detallado de un sistema de software" (Räihä, 2010). Un proceso de diseño de *software* es un puente entre lo que el software necesita hacer ERS, y la implementación del software código (Yau & Tsai, 2018).

Durante el desarrollo de cualquier sistema informático, existen características similares dentro del código para la elaboración de cajas de texto, menús, botones, entre otros. "Muchas de estas características, suelen repetirse incontables veces a lo largo del desarrollo de un proyecto y generan un código realmente extenso y difícil de mantener" (Al-Hawari, 2022).

Para dar solución a estos problemas se crearon los patrones de diseño. El término de patrón surgió por primera vez en 1977 por el arquitecto Christopher Alexander, quien los describió como "una solución a un problema recurrente de nuestro entorno, la cual puede ser usada un millón de veces y de forma diferente" (Guerrero et al., 2013).

Debido al gran número de patrones de diseño desarrollados hasta la actualidad, es necesario agruparlos en categorías dependiendo de sus funcionalidades o características similares. Entre estas categorías se encuentran los patrones arquitectónicos, los cuales son encargados de definir la estructura básica de un sistema, es decir, son una plantilla base sobre la cual se construye una aplicación.

El patrón que será objeto principal de estudio durante este trabajo; es aquel conocido como Modelo-Vista-Controlador también llamado MVC. Este surge de la necesidad de crear un software robusto, que facilite el mantenimiento y reutilización de código a lo largo de todo el ciclo de vida del sistema. "Trygve Reenskaug fue el primero en describir este patrón en el año 1979, sin embargo, no sería aplicado hasta 1988 donde se publicó una formalización del concepto MVC desarrollada por Jim Althoff; tras implementarlo con el objetivo de agregar nuevas funcionalidades al lenguaje de programación Smalltak80" (Espitia et al., 2018).

MVC propone dividir la aplicación en tres módulos llamados modelo, vista y controlador. Cada uno de estos, se encuentra claramente diferenciado y tiene una funcionalidad definida. "El modelo almacena representaciones abstractas de la información, que el sistema va a manejar, la vista se encarga de presentar la información a los usuarios finales y el controlador se encarga de dirigir el flujo de la aplicación" (Basc, 2017). Esto quiere decir que cuando un usuario interactúa con el sistema lo hace a través de la vista, después el controlador recibe todas las solicitudes realizadas y las procesa. Durante este procesamiento el controlador interactúa con el modelo, para manipular la información que se requiera y llevar a cabo la solicitud. Una vez finalizado el procesamiento, el controlador informa a la vista que presente los resultados al usuario.

Además de MVC, existen otros patrones que nos ayudan a organizar y estructurar aplicaciones de gran tamaño y complejidad, en esta área se encuentra el patrón de arquitectura de Microservicios. "El patrón de Microservicios se basa en el desarrollo y despliegue de pequeños servicios o aplicaciones que trabajan y se ejecutan de manera independiente" (Lopez & Edgar, 2017), éste se puede definir también como "una colección de pequeños componentes de software que son considerablemente manejables y comprobables" (Petrasch, 2019).

Al igual que el modelo MVC, los Microservicios "nos ayudan a solucionar varios problemas, durante el desarrollo de un sistema informático y proporcionan a este, características de calidad, modularidad y mantenibilidad del software" (Yunanto et al., 2021).

En el estudio realizado, se identificó al patrón estructural Command-Query Responsability Segregation (CQRS) el cual se basa en una separación de la lógica de lectura (consulta) y escritura (comandos) en aplicaciones que trabajan con un amplio volumen de datos. "En el patrón CQRS la ejecución de los comandos, se lleva a cabo mediante componentes diferentes al que ejecutó las consultas, de forma distribuida, solventa los problemas de escalabilidad que una aplicación pueda tener, este concepto fue introducido por Betrand Meyer, quien también sustentó que cada método corresponde a una acción específica o a una consulta inteligente de datos" (Petrasch, 2019), (Yunanto et al., 2021).

En la actualidad, la industria de desarrollo de software está en constante evolución y mejora. Los desarrolladores buscan constantemente formas de mejorar la calidad y velocidad de su trabajo como desarrolladores (Holzinger et al., 2017). Por lo tanto, es importante que estos comprendan las diferentes arquitecturas de software disponibles y elijan la que mejor se adapte a sus necesidades. Este estudio se enfoca en el popular patrón Modelo-VistaControlador (MVC) y realiza una comparación detallada de sus ventajas y desventajas en relación con otras arquitecturas de software como CQRS y Microservicios. El objetivo de este artículo de revisión es analizar el grado de influencia que tiene la implementación del patrón de diseño MVC sobre la seguridad, interoperabilidad y usabilidad de una solución tecnológica durante todo su ciclo de vida.

2. MATERIALES Y MÉTODOS

Analizada la metodología de revisión documental, y la información de los artículos científicos en diferentes bases bibliográficas que están basados en temas de seguridad, interoperabilidad y usabilidad en sistemas informáticos, se alcanzó a evidenciar el impacto positivo del patrón Modelo-Vista-Controlador (MVC), durante el ciclo de vida de un sistema informático.

Para obtener una mejor comprensión del contenido del presente trabajo se plantearon cuatro objetivos específicos:

- Definir y clasificar los patrones de diseño más comunes para obtener un fundamento teórico que sustente las comparaciones entre estos.
- Evaluar la seguridad, interoperabilidad y usabilidad de aplicaciones de software que han implementado el patrón de diseño Modelo Vista Controlador (MVC) para definir cuáles son sus ventajas y desventajas frente a otros patrones de diseño.
- Medir la satisfacción de los desarrolladores en soluciones tecnológicas, que implementen el patrón de diseño Modelo Vista Controlador para conocer la contribución que este genera en la comunidad de desarrolladores de software.

Se realizó una revisión bibliográfica de fuentes primarias y secundarias e información extraída de páginas web especializadas, en el desarrollo de software, dichas páginas web son reconocidas por la comunidad como referentes tecnológicos.

2.1. Ejemplos de software que implementan MVC

El patrón modelo vista controlador (MVC) es utilizado en la actualidad en diversas aplicaciones ya que permite la separación clara de las diferentes responsabilidades en el desarrollo de una aplicación, lo que facilita la gestión del código, la escalabilidad y la reutilización de componentes.

Un ejemplo de la implementación de este patrón de diseño es “una aplicación web que ofrece un sistema de examen en línea. Esta aplicación fue diseñada con el propósito de evaluar de manera eficiente los conocimientos de los estudiantes. Permitiendo ahorrar el tiempo, proporcionar resultados inmediatos y ayudar a los estudiantes a elegir el tipo de examen que desea rendir a partir de un banco de preguntas o a su vez exámenes preparados” (Liu & Wang, 2012). Para mejorar la calidad de la enseñanza, se desarrolló un banco de preguntas bilingüe y un sistema de examen en línea para estudiantes de medicina y jóvenes médicos utilizando el lenguaje de modelado unificado (UML) y el patrón de diseño Modelo-Vista-Controlador (MVC). UML es un lenguaje de modelado visual, que representa una colección de las mejores prácticas de ingeniería para modelar sistemas grandes y complejos. El sistema de examen en línea se desarrolló utilizando el lenguaje PHP y la base de datos MySQL, con el marco Think PHP en el lado del servidor y el marco JavaScript Jquery en el lado del cliente. Los resultados al implementar el patrón de diseño MVC fueron satisfactorios ya que el modelo se habría encargado de la gestión y el acceso a la base de datos, la vista habría mostrado la información al usuario y recogido su entrada, y el controlador habría coordinado la interacción entre la vista y el modelo automatizando la eficiencia en la toma de exámenes de los estudiantes.

Un ejemplo de la implementación de este patrón de diseño es una aplicación web que ofrece un sistema de examen en línea. Esta aplicación, fue diseñada con el propósito de evaluar de manera eficiente los conocimientos de los estudiantes. “Permitiendo ahorrar tiempo, proporcionar resultados inmediatos y ayudar a los estudiantes a elegir el tipo de examen que desea rendir a partir de un banco de preguntas o a su vez exámenes preparados” (Liu & Wang, 2012). Así, para mejorar la calidad de la enseñanza, se desarrolló un banco de preguntas bilingüe y un sistema de examen en línea para estudiantes de medicina y jóvenes médicos utilizando el lenguaje de modelado unificado (UML) y el patrón de diseño Modelo-Vista-Controlador (MVC). UML es un lenguaje de modelado visual que representa una colección de las mejores prácticas de ingeniería para modelar sistemas grandes y complejos. El sistema de examen en línea se desarrolló utilizando el lenguaje PHP y la base de datos MySQL, con el marco ThinkPHP en el lado del servidor y el marco JavaScript Jquery en el lado del cliente. Los resultados al implementar el patrón de diseño MVC fueron satisfactorios ya que el modelo se habría encargado de la gestión y el acceso a la base de datos, la vista habría mostrado la información al usuario y recogido su entrada, y el controlador habría coordinado la interacción entre la vista y el modelo automatizando la eficiencia en la toma de exámenes de los estudiantes.

El mundo de los videojuegos crece cada vez más en la actualidad. Por tal razón es necesario, la implementación de un patrón de diseño en los videojuegos, esto ayuda a separar el código del juego con el renderizado, motivando a los desarrolladores a crear aplicaciones escalables, que pueda tener una transición a nuevas tecnologías o plataformas diferentes. Existe otro ejemplo de la implementación de la modelo vista controlador que se implementó en cinco proyectos de software basados en videojuegos: Time Breaker, Frontline, T.W.T.P.B., Gears of Love y Hero. Por medio

de los cuales se realizó, una investigación comparativa basada en pruebas de rendimiento y calidad. “Los resultados encontraron tres variantes del patrón MVC: una en la que la vista y el controlador están mezclados, otra en la que el controlador tiene una capa de vista más relajada y otra en la que el controlador tiene una capa de vista estricta. La variante con una capa de vista estricta se consideró la mejor en términos de calidad del software” (Ollsson et al., 2019).

3. RESULTADOS Y DISCUSIÓN

3.1. Ventajas y desventajas de utilizar MVC

Con base a lo antes citado, el modelo MVC separa la lógica de presentación de la lógica de negocio de una aplicación, y en relación con otros patrones analizados se pueden destacar los siguientes puntos:

Ventajas

- Reutilización del código: “debido a que MVC se basa en componentes que separan la parte visual de la lógica del negocio” (Yair & Sánchez, 2020).
- Muy utilizado: “existe un gran número de aplicaciones basadas en MVC y de gran relevancia como Java Swing y Microsoft” (Basc, 2017).
- Facilidad de testing: “el patrón MVC presenta una mayor facilidad de realización de pruebas que los Microservicios debido a su separación de responsabilidades (presentación y lógica de negocio) o intereses” (Yair & Sánchez, 2020).
- Mejor mantenibilidad a futuro. En relación con CQRS debido a que este puede ser más costoso por la infraestructura adicional para el manejo de separación de lectura y escritura.

Desventajas

- Complejidad de aplicación: requiere mayor conocimiento por parte de los desarrolladores.
- Debido a la separación de componentes se puede generar un amplio número de archivos.
- Requiere la existencia de una arquitectura inicial (Basc, 2017).

3.2. Nivel de satisfacción de los desarrolladores con MVC

Durante la presente investigación, se encontró un estudio realizado, su objetivo describe lo siguiente; “conocer cuáles eran las influencias de implementar un patrón de diseño en la creación de un sistema”. Dicho estudio analiza las respuestas de 206 personas relacionadas con el desarrollo de software y concluye que las personas con mayor experiencia en el desarrollo de software prefieren el uso de un patrón de diseño y no implementar ninguno. “Las personas encuestadas mencionaron que han tenido experiencias positivas con la utilización de patrones de diseño, las cuales les permiten afirmar que los patrones de diseño facilitan el desarrollo de un aplicativo, así como también mejora su flexibilidad y mantenibilidad” (Zhang et al., 2018).

“Otro estudio realizado en 2019 con el objetivo de comparar frameworks para el desarrollo back-end de aplicaciones web concluye que las mejores opciones para crear soluciones tecnológicas empresariales de gran escala son Spring y Django” (Kaluža et al., 2019). Es importante mencionar que Spring implementa el patrón MVC dentro de su arquitectura (Spring, 2022), mientras que Django implementa una modificación llamada MTV, sin embargo, “conserva la misma lógica fundamental del patrón modelo vista controlador” (Foundation, 2017).

Otro factor importante para tomar en cuanta es la popularidad de las tecnologías utilizadas por la comunidad de desarrolladores a nivel mundial. “Stack Overflow presentó en su última encuesta una lista de las tecnologías web más utilizadas” (Stack Overflow, 2022). Tomando en cuenta esta lista e investigando en la documentación oficial de cada tecnología, se puede afirmar que la mayoría de los frameworks populares utilizan MVC o alguna variación de este como arquitectura. “Entre los casos más populares para el desarrollo front-end se encuentra Angular el cual según un desarrollador de la empresa internacional DigitalOcean presenta un patrón MVVM que es una variación del patrón modelo vista controlador” (DigitalOcean, 2019).

Otro factor importante para tomar en cuanta es la popularidad de las tecnologías utilizadas por la comunidad de desarrolladores a nivel mundial. StackOverflow presentó en su última encuesta una lista de las tecnologías web más utilizadas (Stack Overflow, 2022). Tomando en cuenta esta lista e investigando en la documentación oficial de cada tecnología se puede afirmar que la mayoría de los frameworks populares utilizan MVC o alguna variación de este como arquitectura. “Entre los casos más populares para el desarrollo front-end se encuentra Angular el cual según un desarrollador de la empresa internacional DigitalOcean presenta un patrón MVVM que es una variación del patrón modelo vista controlador” (DigitalOcean, 2019).

Después de revisar varios foros y discusiones, se observó que la gran mayoría pone en buenas condiciones la implementación del MVC. Algunos usuarios comentan que el patrón MVC puede ser muy útil para mantener una

separación clara entre la lógica del negocio y la interfaz de usuario, lo que les ha permitido lograr una mayor escalabilidad y reutilización de código. Otros usuarios señalan que la implementación de MVC puede ser compleja y llevar a un código complicado y difícil de mantener, especialmente si no se tiene una experiencia en el uso del patrón de diseño. Algunos recomiendan utilizar variantes del patrón MVC, como el patrón MVP o MVVM, para abordar ciertas limitaciones del enfoque tradicional. Así, se puede argumentar que la implementación exitosa de MVC en una aplicación dependerá de varios factores, como el tamaño y complejidad del proyecto, la experiencia del equipo de desarrollo y la comprensión detallada del patrón y sus mejores prácticas.

CONCLUSIONES

El estudio de los patrones de diseño en la programación de software es complicado para desarrollar aplicaciones de calidad, eficientes y mantenibles. A través de varios estudios e investigacines, se han identificado los patrones de diseño más comunes.

En particular, el patrón de diseño Modelo Vista Controlador (MVC) ha sido evaluado en términos de seguridad, interoperabilidad y usabilidad, y se han identificado sus ventajas y desventajas en comparación con otros patrones de diseño como CQRS. Los resultados indican que MVC es una arquitectura sólida y escalable, pero puede ser compleja de implementar en proyectos pequeños.

Además, se ha medido la satisfacción de los desarrolladores en soluciones tecnológicas que implementan el patrón MVC, y se ha encontrado que su contribución a la comunidad de desarrolladores de software es significativa, ya que permite una mejor organización del código y una mayor facilidad de mantenimiento.

En conclusión, los patrones de diseño son una herramienta fundamental para la programación de software, y el patrón MVC en particular puede ser una excelente opción para proyectos de mediana y gran escala. Es importante que los desarrolladores estén familiarizados con estos patrones y sus ventajas y desventajas para poder tomar buenas decisiones al diseñar y desarrollar aplicaciones de software.

Declaración de conflicto de intereses

Los autores declaran que no existe ningún conflicto de interés potencial dentro de esta investigación, autoría y/o publicación de este artículo.

REFERENCIAS

Al-Hawari, F. (2022). Software design patterns for data management features in web-based information systems. *Journal of King Saud University - Computer and Information Sciences*, *34*(10). https://doi.org/10.1016/j.jksuci.2022.10.003.

Basc, E. (2017). El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing. *Acta Nova*, *2*(Mvc), 493–507. https://hdl.handle.net/11042/2743.

DigitalOcean. (2019). *Angular MVC - A Primer*. https://www.digitalocean.com/community/tutorials/angular-angular-mvc-primer.

Espitia, N., Armao, O., & Carbajo, J. (2018). *República Bolivariana de Venezuela*.

Foundation, D. S. (2017). *FAQ: General*. Django Appears to Be a MVC Framework, but You Call the Controller the "View", and the View the "Template". How Come You Don't Use the Standard Names? https://docs.djangoproject.com/en/4.1/faq/general/#django-appears-to-be-amvc-framework-but-you-call-the-controller-the-view-and-the-view-thetemplate-how-come-you-don-t-use-the-standard-names.

Guerrero, C. A., Suárez, J. M., & Gutiérrez, L. E. (2013). Patrones de diseño GOF (the gang of four) en el contexto de procesos de desarrollo de aplicaciones orientadas a la web. *Informacion Tecnologica*, *24*(3). https://doi.org/10.4067/S0718-07642013000300012.

Holzinger, A., Struggl, K. H., & Debevc, M. (2017). Applying Model-View-Controller (MVC) in design and development of information systems: An example of smart assistive script breakdown in an e-Business application. In *Proceedings of the International Conference on e-Business (ICE-B 2010)*, 63–68. https://doi.org/10.5220/0002980900630068.

Kaluža, M., Kalanj, M., & Vukelić, B. (2019). A comparison of back-end frameworks for web application development. *Zbornik Veleučilišta u Rijeci*, *7*(1), 317–332. https://doi.org/10.31784/zvr.7.1.10.

Liu, C., & Wang, K. (2012). An online examination system based on UML modeling and MVC design pattern. In *Proceedings - 2012 International Conference on Control Engineering and Communication Technology, ICCECT 2012*, 815–817. https://doi.org/10.1109/ICCECT.2012.189.

Lopez, D., & Edgar, M. (2017). Arquitectura de Software basada en Microservicios para Desarrollo. *Séptima Conferencia de Directores de Tecnología de Información, TICAL 2017 Gestión de Las TICs Para La Investigación y La Colaboración, San José, Del XX al XX de Julio de 2017*, 5–7.

Ollsson, T., Toll, D., Wingkvist, A., & Ericsson, M. (2019). Evolution and Evaluation of the Model-View-Controller Architecture in Games. *Proceedings - 4th International Workshop on Games and Software Engineering, GAS 2015*, 8–14. https://doi.org/10.1109/GAS.2015.10.

Petrasch, R. (2019). Transformation of state machines for a microservice-based event-driven architecture: A proof-of-concept. *Advances in Intelligent Systems and Computing*, *769*, 327–336. https://doi.org/10.1007/978-3-319-93692-5_32.

Räihä, O. (2010). A survey on search-based software design. *Computer Science Review*, *4*(4), 203–249. https://doi.org/10.1016/J.COSREV.2010.06.001.

Räihä, O. (2018). A survey on search-based software design. *Computer Science Review*, *4*(4), 203–249. https://doi.org/10.1016/j.cosrev.2010.06.001.

Spring, (2022). *Spring MVC Tutorial*. https://www.baeldung.com/spring-mvc-tutorial

Stack Overflow. (2022). *Stack Overflow Developer Survey Result*. Stack Overflow. https://survey.stackoverflow.co/2022/#technology-most-popular-technologies.

Yair, P., & Sánchez, C. (2020). Implementación del patrón arquitectónico MVC en aplicaciones web para la arquitectura del software del sistema de capellanía de la UM. *Anuario2020*, *1*(1), 112–120.

Yau, S. S., & Tsai, J. J. P. (2018). A Survey of Software Design Techniques. *IEEE Transactions on Software Engineering*, *SE-12*(6), 713–721. https://doi.org/10.1109/TSE.1986.6312969.

Yunanto, A. A., Hardiansyah, F. F., Putra, A. A. A., Rasyid, M. B. A., & Arifiani, S. (2021). Development of sandbox components with microservices architecture and design patterns in games. *Procedia Computer Science*, *197*, 354–361. https://doi.org/10.1016/j.procs.2021.12.150.

Zhang, C., Wang, F., Xu, R., Li, X., & Yang, Y. (2018). A quantitative analysis of survey data for software design patterns. *ACM International Conference Proceeding Series*, 48–55. https://doi.org/10.1145/2627508.2627516.